**ЛАБОРАТОРНА РОБОТА №3**

**Тема:**Перевантаження операцій класу

**Мета:**ознайомитись зі способами перевантаження операцій та навчитись використовувати їх при роботі з об’єктами.

**Завдання 1.** Ознайомитись зі способами перевантаження операцій у С++.

Ознайомився зі способами перевантаження операцій у С++.

**Завдання 2.** Проаналізувати приклад програми, яка оголошує клас та перевантажує низку операцій над точками у тривимірному просторі. Визначити результат її  роботи.

Проаналізував приклад програми, яка оголошує клас та перевантажує низку операцій над точками у тривимірному просторі. Визначити результат її  роботи.

**Завдання 3.** В класі Int, який розроблений в завданні №1 лабораторної роботи №1,  перевизначте чотири цілочисельні арифметичні операції («+», «-», «\*» , «/») так, щоб їх можна було використовувати для операцій з об'єктами класу Int. Якщо результат будь-якої з операцій виходить за межі типу int (в 32-бітній системі), що може мати значення від 2 147 483 648 до -2 147 483 648, то операція повинна послати повідомлення про помилку і завершити програму. Такі типи даних корисні там, де помилки можуть бути викликані арифметичним переповненням, яке неприпустимо.

Напишіть програму для перевірки цього класу.

*Підказка: для полегшення перевірки переповнення виконуйте обчислення з використанням типу long double.*

**Код програми**

#include <iostream>

using namespace std;

class **Int** {

private:

int Inte;

public:

**Int**(long double a){

if(a>2147483684 || a<-2147483684){

cout<<"ERROR"<<endl;

}

else{

Inte = (int)a;

}

}

void **setInt**(int \_Int) {

Inte = \_Int;

}

int **getInt**() {

return Inte;

}

void **set0**() {

Inte = 0;

}

int **sum**(int num1, int num2)

{

return num1 + num2;

}

int operator +(Int& a)

{

return Inte+a.Inte;

}

int operator -(Int& a)

{

return Inte-a.Inte;

}

int operator \*(Int& a)

{

return Inte\*a.Inte;

}

double operator /(Int& a)

{

return ((double (Inte))/a.Inte);

}

};

int **main**()

{

Int a(29), b(27);

Int c = a+b;

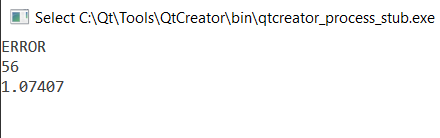
Int d(9999999999);

cout<<c.getInt()<<endl<<a/b<<endl;

return 0;

}

Вивід програми



**Завдання 4.** Для заданого варіанта індивідуального завдання виконати перевантаження операцій для зручності роботи з об’єктами. При необхідності оголосіть певні операторні функції  друзями класу.

**Індивідуальне завдання:**

Створити клас – двозв’язний список. Визначити необхідні конструктори, деструктор. Перевантажити потокові операції введення і виведення, операції + (додавання елемента в початок списку), += (додавання елемента в кінець списку), – (вилучення заданого елемента зі списку), ++ (перехід до наступного елемента), -- (перехід до попереднього елемента).

Код програми

#include <iostream>

using namespace std;

struct **Elem**

{

int data;

Elem \* next, \* prev;

};

class **List**

{

Elem \* Head, \* Tail;

int Count;

public:

**List**();

~**List**();

int **GetCount**();

Elem\* **GetElem**(int);

void operator +(int n);

void operator +=(int n);

void operator - (int pos);

void **Print**();;

void operator<<(int pos);

};

List::**List**()

{

Head = Tail = nullptr; //список пустий

Count = 0;

}

List::~**List**()

{

}

void List::**Print**()

{

if(Count != 0)

{

Elem \* temp = Head;

cout << "( ";

while(temp->next != nullptr)

{

cout << temp->data << ", ";

temp = temp->next;

}

cout << temp->data << " )\n";

}

}

int List::**GetCount**()

{

return Count;

}

void List::operator<<(int pos){

if(pos < 1 || pos > Count)

{

cout << "Incorrect position !!!\n";

return;

}

Elem \* temp;

if(pos <= Count / 2)

{

temp = Head;

int i = 1;

while(i < pos)

{

temp = temp->next;

i++;

}

}

else

{

temp = Tail;

int i = 1;

while(i <= Count - pos)

{

temp = temp->prev;

i++;

}

}

cout << pos << " element: ";

cout << temp->data << endl;

}

void List::operator+(int n)

{

Elem \* temp = new Elem;

temp->prev = nullptr;

temp->data = n;

temp->next = Head;

if(Head != nullptr)

Head->prev = temp;

if(Count == 0)

Head = Tail = temp;

else

Head = temp;

Count++;

}

void List::operator+=(int n){

Elem \* temp = new Elem;

temp->next = nullptr;

temp->data = n;

temp->prev = Tail;

if(Tail != nullptr)

Tail->next = temp;

if(Count == 0)

Head = Tail = temp;

else

Tail = temp;

Count++;

}

void List::operator - (int pos)

{

if(pos == 0)

{

cout << "Input position: ";

cin >> pos;

}

if(pos < 1 || pos > Count)

{

cout << "Incorrect position !!!\n";

return;

}

int i = 1;

Elem \* Del = Head;

while(i < pos)

{

Del = Del->next;

i++;

}

Elem \* PrevDel = Del->prev;

Elem \* AfterDel = Del->next;

if(PrevDel != nullptr && Count != 1)

PrevDel->next = AfterDel;

if(AfterDel != nullptr && Count != 1)

AfterDel->prev = PrevDel;

if(pos == 1)

Head = AfterDel;

if(pos == Count)

Tail = PrevDel;

delete Del;

Count--;

}

int **main**()

{

List L;

int a[10] = {0,1,2,3,4,5,6,7,8,9};

for(int i = 0; i < 10; i++)

if(i % 2 == 0)

L +a[i];

else

L+=a[i];

L-3;

cout << "List L:"<<endl;

L.Print();

cout << endl;

cout << "List L:"<<endl;

L.Print();

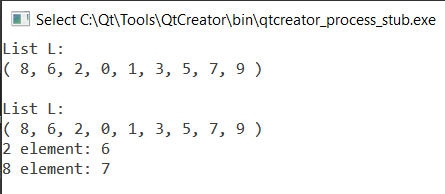
L<<2;

L<<8;

return 0;

}

Вивід програми



**Висновок:**ознайомився зі способами перевантаження операцій та навчивсь використовувати їх при роботі з об’єктами.